**React JS :**

**Components:**

**Components are the building blocks of a React application. They are reusable, independent pieces of UI that can be composed together to create complex interfaces. Components can be either functional or class-based.**

* **Class Components**
* **Functional Components**
* **Higher Order Components(HOC)**
* **Nested Components**

**Functional Components:**

Functional components in React are simple JavaScript functions that accept props as input and return React elements to render UI. They are stateless, meaning they don't manage their own state, and are typically used for presenting UI based on props received from parent components. Functional components have become more powerful with the introduction of React hooks, allowing them to use state and other React features.

import React,{useState} from 'react';

const FunctionalComponent = () => {

    const [count, setCount] = useState(0);

    const incrementCount = () => {

      setCount(count + 1);

    };

    const decrementCount = () => {

      setCount(count - 1);

    };

    const reset = () => {

        setCount(0);

    }

  return (

    <div style={{backgroundColor:'rgb(199, 121, 121)', padding:'15px', margin:'10px 0'}}>

        <center>

        <h1 style={{fontWeight:'bold', fontSize : '30px', paddingBottom:'10px',textDecoration:'underline' }} >Functional Component Counter</h1>

        <h2><span style={{fontWeight:'bolder',fontSize:'30px',paddingRight:'5px'}}>Count</span>: <span style={{fontWeight:'bold',fontSize:'30px',color:'blueviolet'}}> {count}</span></h2>

    <button onClick={incrementCount}>Increment</button>

    <button onClick={decrementCount}>Decrement</button>

    <button onClick={reset}>Reset</button>

        </center>

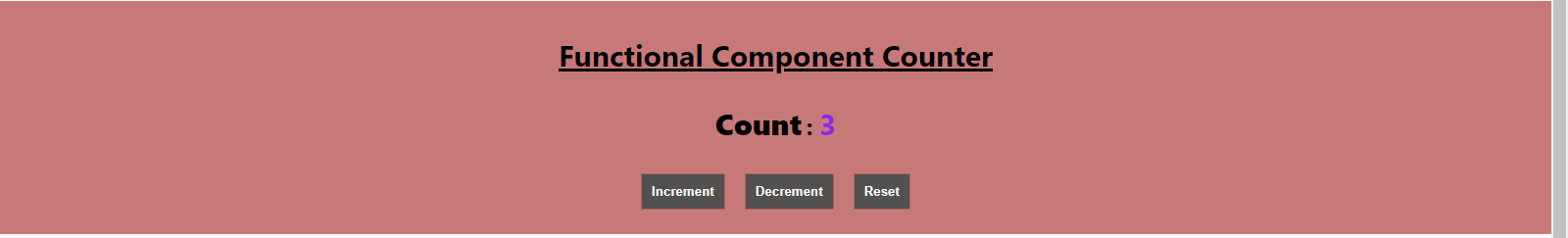
  </div>

  )

}

export default FunctionalComponent

This is an example of Functional Component in this example I have created a counter stating increment, decrement and reset of the count. It is an arrow function in which we have used “Use State” Hook and we have took count as set State and we wrote a increment and decrement and reset function for the Use State “Count”.



**Nested Components:**

Nested components in React refer to the practice of composing components within other components, forming a hierarchical structure. This approach enables developers to break down complex UIs into smaller, reusable, and manageable parts. Each nested component can encapsulate its own functionality, state, and rendering logic, promoting code reusability and maintainability. By nesting components, developers can create modular, well-organized, and easily understandable React applications.

import React, { useState } from 'react';

const NestedComponent = () => {

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const handleSubmit = () => {

    setIsLoggedIn(true);

  };

  return (

    <div style={{backgroundColor:'beige',padding:'40px', margin:'10px 0', border:'3px solid red' }}>

      {isLoggedIn ? (

        <div>

          <center>

          <h1 style={{color:'forestgreen'}}>Login Successful!</h1>

          </center>

        </div>

      ) : (

        <center>

        <form onSubmit={handleSubmit}>

          <input type="text" placeholder="Username" /> <br />

          <input type="password" placeholder="Password" /> <br />

          <button type="submit">Submit</button>

        </form>

        </center>

      )}

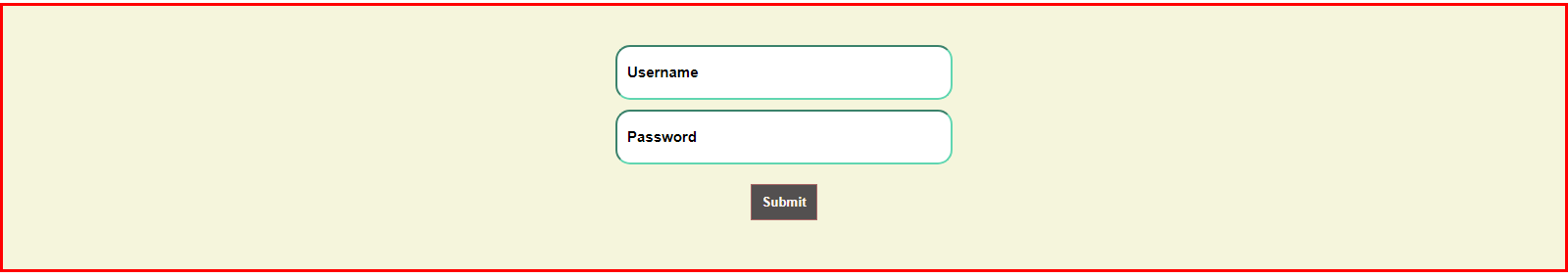
    </div>

  );

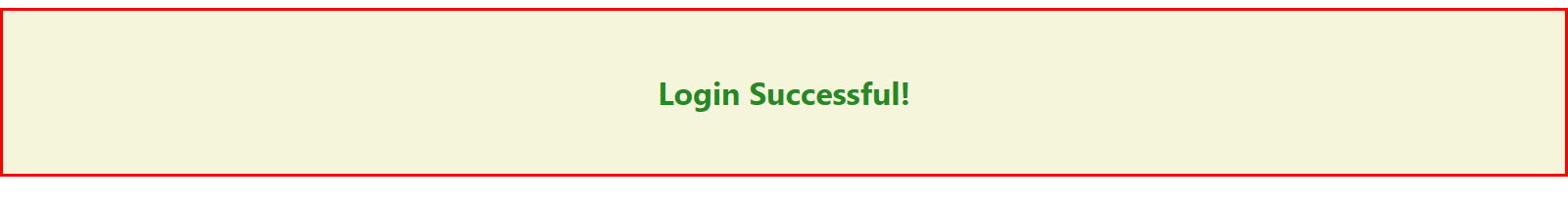
};

export default NestedComponent;

This an example of Nested Component Where we have Used Two Components where <h1>Login Successful</h1> is one Component and <form></form> is another component here if we fill the details in Form component and click on submit it will redirect to the other component and shows Login Successful



After Clicking Submit Button we will get Login Successful



We will get the Output like This.

**Higher Order Components (HOC’s):**

Higher-Order Components (HOCs) in React are functions that take a component and return a new component with enhanced functionality. They enable code reuse, abstraction of logic, and separation of concerns by allowing common functionality to be applied to multiple components. HOCs enhance composability and maintainability by promoting a modular and reusable design pattern.

import React, { Component } from 'react';

import { ToastContainer, toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

function withForm(Component) {

  return class extends Component {

    state = { formData: {} };

    handleChange = (e) => {

      const { name, value } = e.target;

      this.setState(prevState => ({

        formData: { ...prevState.formData, [name]: value }

      }));

    };

    handleSubmit = (e) => {

        e.preventDefault();

      const { formData } = this.state;

      toast.success('Username: ' + formData.username + '\n Password: ' + formData.password);

    };

    render() {

      return (

        <Component

          formData={this.state.formData}

          handleChange={this.handleChange}

          handleSubmit={this.handleSubmit}

          {...this.props}

        />

      );

    }

  };

}

class MyForm extends Component {

  render() {

    const { handleChange, handleSubmit } = this.props;

    return (

        <center>

            <form onSubmit={handleSubmit}>

                <ToastContainer />

                <input type="text" name="username" onChange={handleChange} placeholder="Username" /> <br />

                <input type="password" name="password" onChange={handleChange} placeholder="Password" /> <br />

                <button type="submit">Submit</button>

            </form>

        </center>

    );

  }

}

const FormComponent = withForm(MyForm);

export default FormComponent;

In this HOC we have wrapped our form into HOC With Form here in this we have used Toast message to show the form inputs we have entered in toast after clicking the submit button.

